单片机汇编&C51程序结构

信息工程学院　　陈梅

单片机编程语言

1．汇编语言

- 特点：占用空间小，运行速度快；但可读性差，离不开具体的硬件。

2．高级语言——目前公认Keil C51

- 特点：可读性好，直观、易懂、易学；不受具体“硬件”的限制。

- 尽管目前多数设计人员使用C51来进行程序开发，但在对程序的空间和时间要求较高的场合，汇编语言仍必不可少。
- 在很多需要直接控制硬件且对实时性要求较高的场合，则更是非用汇编语言不可。
- 在这种场合下，可使用C语言和汇编语言混合编程。

- 因此，掌握汇编语言并能进行程序设计，是学习和掌握单片机程序设计的基本功之一。

汇编语言的两种基本语句：指令语句和伪指令语句。

（1）指令语句：

每一指令语句在汇编时都产生一个指令代码（机器代码），执行该指令代码对应着机器的一种操作。

（2）伪指令语句：

“伪”体现在汇编后，伪指令没有相应的机器代码产生。

只有在汇编前的源程序中才有伪指令。伪指令不属于指令系统中的汇编语言指令，是控制汇编（翻译）过程的一些控制命令。

汇编程序结构

常见结构：

```
        ORG    0000H                ；不可省
        LJMP   MAIN
        ****   EQU/BIT   ****
        ORG    0100H
MAIN:   ……
        ……
SUB:    ……
        ……
****:   DB/DW   **, ……
        END                         ；不可省
```

1. 格式：ORG　十进制/十六进制数　；汇编起始地址命令

- 在一个源程序中，可多次用ORG指令，地址必须由小到大排列，且不能交叉、重叠。

这种顺序正确

```
ORG     2000H
……
ORG     2500H
……
ORG     3000H
……
```

这种顺序错误

```
ORG     2500H
……
ORG     2000H
……
ORG     3000H
……
```

2.　格式：　END　；汇编终止命令

- 源程序结束标志，终止源程序的汇编工作。
- 整个源程序中只能有一条END命令，且位于程序的最后。

3．格式：标号 EQU 数值/符号 ；标号赋值命令

- 用于给标号赋值。其标号值在整个程序有效。
- 一般放在程序开始处。

例： LED EQU P1

4．格式： 字符名称 BIT 位地址操作数 ；位定义命令

- 用于给字符名称赋绝对位地址或符号地址。

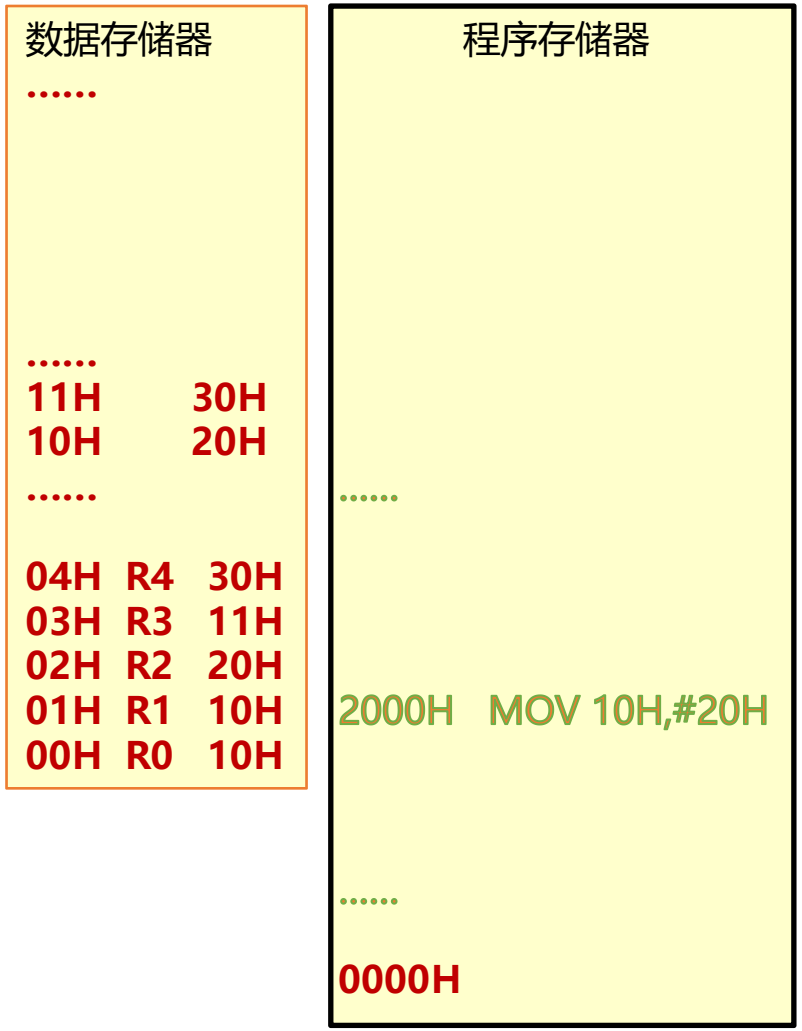
例： P10 BIT P1.0

WR BIT P3.6

例：

```
        ORG 0000H
        LJMP START
        ORG 2000H
COUNT   EQU  10H
START:  MOV 10H,#20H        ;(10H)=20H
        MOV 11H,#30H        ;(11H)=30H
        MOV R0,#10H         ;R0=10H
        MOV R1,#COUNT       ;R1=10H
        MOV R2,COUNT        ;R2=20H
        MOV R3,#COUNT+1     ;R3=11H
        MOV R4,COUNT+1      ;R4=30H
        SJMP $
        END
```

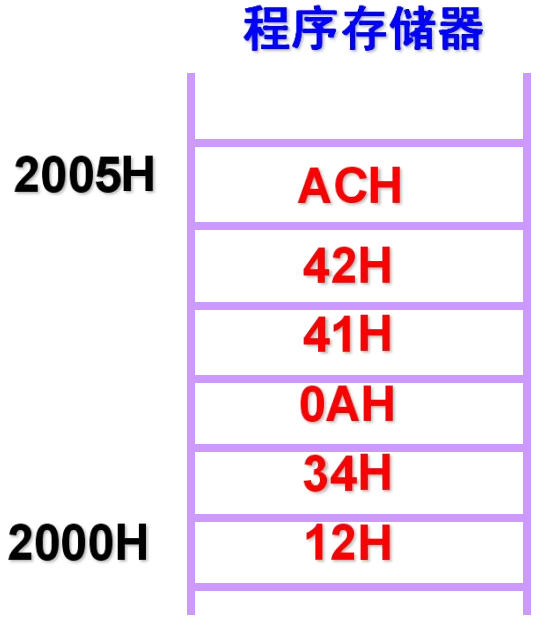
执行后：



5. 格式：DB　字节常数或ASCII字符

功能：用于从指定地址开始在程序存储器连续单元中定义字节数据。

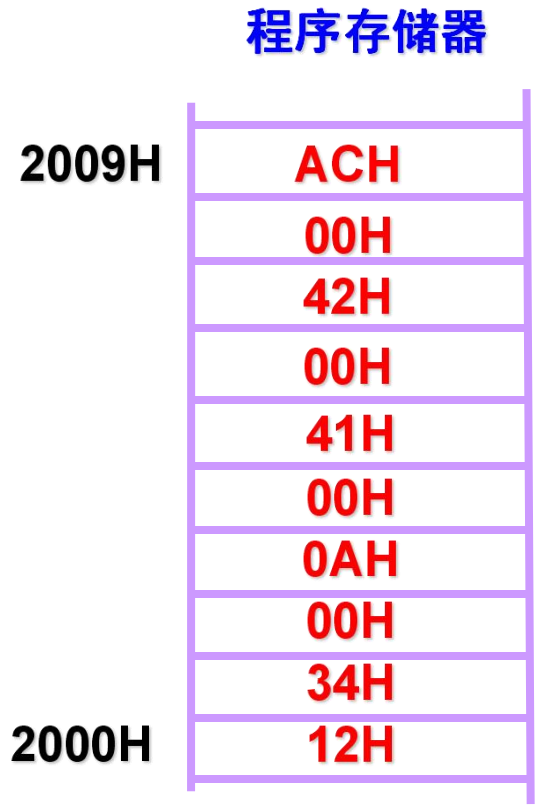
例：　　ORG　2000H
　　DB　　12H，34H，10
　　DB　　"A"，"B"，0ACH



6. 格式：DW　字常数或ASCII字符

功能：用于从指定地址开始在程序存储器的连续单元中定义字数据。

例：　　ORG　2000H
　　DW　　1234H，10，
　　DW　　"A"，"B"，0ACH



7．格式： DS 表达式 ;定义存储区命令

功能：从指定地址开始，保留指定数目的字节单元作为存储区，供程序运行使用。

例： ORG 2000H

DS 08H

DB 30H,8AH ;则30H从2008H单元开始存放



DB、DW和DS命令只能对程序存储器有效，不能对数据存储器使用。

SAMP.ASM

```
 1        ORG     0000H
 2        LJMP    START
 3        ORG     0100H
 4  TAB:  DB      12H,34H,10
 5        DB      "A","B",0ACH
 6        DW      1234H,10
 7        DW      "A","B",0ACH
 8        DS      08H
 9        DB      "#"
10  COUNT EQU     10
11
12        ORG     0160H
13  START:MOV     10H,#20H
14        MOV     11H,#30H
15        MOV     R0,#10H
16        MOV     R1,#COUNT
17        MOV     R2,COUNT
18        MOV     R3,#COUNT+1
19        MOV     R4,COUNT+1
20        SJMP    $
21        END
22
```

程序存储器

```
.....

0160H   MOV 10H,#20H
0118H       23H
            .....
0110H       00H
            ACH
            00H
            42H
            00H
            41H
            00H
            0AH
            00H
            34H
0106H       12H
            ACH
            "B"
            "A"
            0AH
            34H
0100H       12H
.....
0000H       LJMP
```

Memory 1

Address: C:0000H

```
C:0x0000: 02 01 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x00A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x00C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x00E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0100: 12 34 0A 41 42 AC 12 34 00 0A 00 41 00 42 00 AC 00 00 00 00 00 00 00 00 23 00
C:0x0120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x0160: 75 10 20 75 11 30 78 10 79 0A AA 0A 7B 0B AC 0B 80 FE 00 00 00 00 00 00 00 00
C:0x0180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x01A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x01C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C:0x01E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Call Stack + Locals | Memory 1

程序存储器

```
.....

0160H   MOV 10H,#20H
0118H       23H
            .....
0110H       00H
            ACH
            00H
            42H
            00H
            41H
            00H
            0AH
            00H
            34H
0106H       12H
            ACH
            “B”
            “A”
            0AH
            34H
0100H       12H
.....
0000H       LJMP
```

Disassembly

```
C:0x00FD    00        NOP
C:0x00FE    00        NOP
C:0x00FF    00        NOP
C:0x0100    12340A    LCALL     C:340A
C:0x0103    4142      AJMP      C:0242
C:0x0105    AC12      MOV       R4,0x12
C:0x0107    3400      ADDC      A,#0x00
C:0x0109    0A        INC       R2
C:0x010A    00        NOP
C:0x010B    4100      AJMP      C:0200
C:0x010D    4200      ORL       0x00,A
C:0x010F    AC00      MOV       R4,0x00
C:0x0111    00        NOP
C:0x0112    00        NOP
C:0x0113    00        NOP
C:0x0114    00        NOP
C:0x0115    00        NOP
C:0x0116    00        NOP
C:0x0117    00        NOP
C:0x0118    23        RL        A
C:0x0119    00        NOP
```

Disassembly

```
C:0x015D    00        NOP
C:0x015E    00        NOP
C:0x015F    00        NOP
    13: START:  MOV 10H,#20H
C:0x0160    751020    MOV       0x10,#0x20
    14:                 MOV 11H,#30H
C:0x0163    751130    MOV       0x11,#0x30
    15:           MOV R0,#10H
C:0x0166    7810      MOV       R0,#0x10
    16:                 MOV R1,#COUNT
C:0x0168    790A      MOV       R1,#0x0A
    17:                 MOV R2,COUNT
C:0x016A    AA0A      MOV       R2,0x0A
    18:                 MOV R3,#COUNT+1
C:0x016C    7B0B      MOV       R3,#0x0B
    19:                 MOV R4,COUNT+1
C:0x016E    AC0B      MOV       R4,0x0B
    20:                 SJMP $
C:0x0170    80FE      SJMP      C:0170
C:0x0172    00        NOP
C:0x0173    00        NOP
```

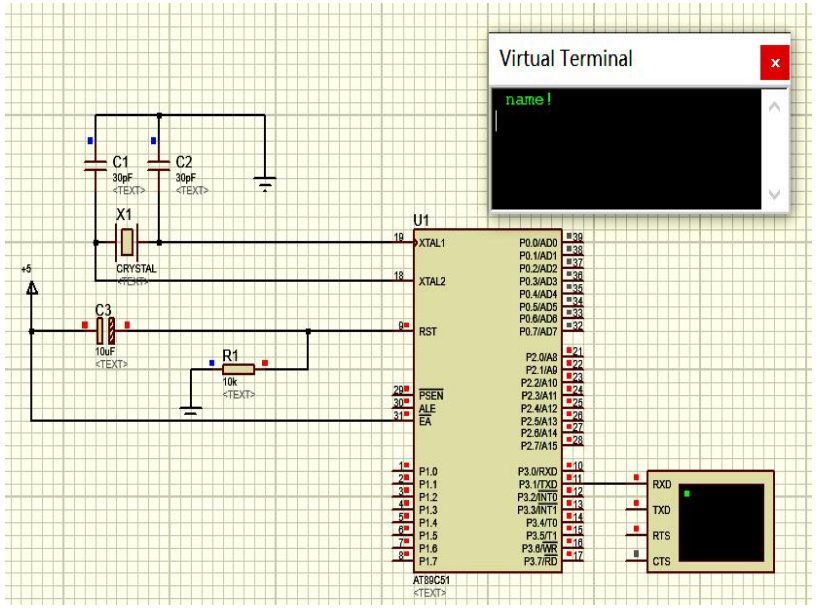
C51程序结构

C51与标准C程序结构几乎完全相同，C51的强大体现在库函数。

（1）标准库函数

STARTUP.A51 | hello.c

```
1  #include <reg52.h>
2  #include <stdio.h>
3
4  void UartInit(void)
5  {
6     SCON=0x40;              //串口工作于方式1发送
7     TMOD=0x20;              //T1工作于方式2用于产生波特率
8     TH1=0xFD;               //晶振为11.0592MHz,波特率为9.6KHz
9     TR1=1;                  //启动定时器
10    TI=1;                   //表示一帧数据发送结束
11 }
12
13 void main(void)
14 {
15    UartInit();
16    printf(" name!\n"); //把输出内容改为你姓名的全拼
17    while(1);
18 }
```



C51的库函数包含：本征函数和非本征函数

- 本征函数：在编译时直接将固定代码插入当前行，不是用CALL指令实现，大大提高了函数访问效率。

C51本征函数共9个，在头文件<intrins.h>中：

- _crol_,_cror_：将char型变量循环左（右）移动指定位数
- _irol_,_iror_：将int型变量循环左（右）移动指定位数
- _lrol_,_lror_：将long型变量循环左（右）移动指定位
- _nop_：相当于插入汇编指令NOP
- _testbit_：相当于汇编指令JBC bitvar测试该位变量并跳转同时清除
- _chkfloat_：测试并返回浮点数状态

C51的库函数包含：本征函数和非本征函数

- 非本征函数：必须由CALL指令调用。一般说的库函数指非本征函数。

  几类重要的库函数：

  - reg51.h或reg52.h专用寄存器文件，包括了所有80C51或80C52的SFR及其定义；
  - absacc.h绝对地址文件，定义了几个宏以确定各存储空间的绝对地址；
  - string.h缓冲区处理函数，包括了复制memcpy、比较memcmp、移动memmove等缓冲区处理函数。
  - stdlib.h动态内存分配函数
  - stdio.h输入输出流函数，通过串口或用户定义的I/O口读写数据，默认是串口。如果要修改，比如改为LCD显示，可修改lib目录中的getkey.c及putchar.c源文件，在库中替换它们即可。

（2）用户自定义函数

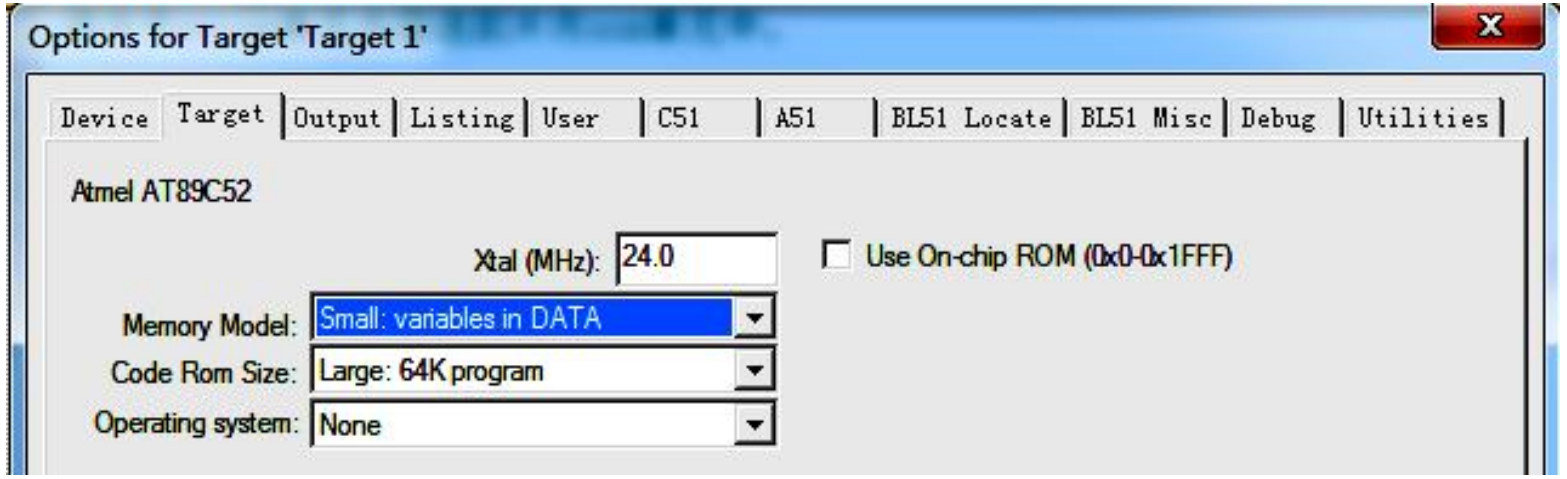
格式：返回值类型　函数名(形参列表) **[编译模式]** [reentrant] [interrupt n] [using n]

```
{
    函数体
}
```

说明：　1、编译模式用来指定函数中局部变量和参数的存储器空间

存储类型	与存储空间的对应关系	编译模式
data	片内RAM 直接寻址区，位于片内RAM的低128字节	SMALL
bdata	片内RAM 位寻址区，位于20H~2FH空间，允许位访问与字节访问	
idata	片内RAM 间接寻址区，全部256字节存储区00H~FFH，由MOV @Ri访问	
pdata	片外RAM的一个分页间接寻址区，占256字节，使用指令MOVX @Ri访问	COMPACT
xdata	片外RAM 间接寻址区，全部64K字节空间，使用MOVX @DPTR访问	LARGE
code	程序存储区，全部64KB空间，使用MOVC @A+DPTR访问	

- 如果在变量定义时省略了存储器类型标识符，编译器会根据当前采取的编译模式自动认定默认的存储类型。



约定：若无特殊声明，一般均为“SMALL编译模式”

编译模式	默认存储分区	特点
SMALL	data	变量在片内RAM，空间小，速度快，适用于小程序
COMPACT	pdata	变量在片外RAM的一页（256B）
LARGE	xdata	变量在片外RAM的64KB，空间大，速度慢

（2）用户自定义函数

格式：返回值类型　函数名(形参列表) [编译模式] [reentrant] [interrupt n] [using n]

```
{
    函数体
}
```

说明：

2、C51使用递归必须用reentrant 声明，表示该函数为可重入函数，因为51单片机的硬件资源有限，C51编译系统不允许太多的程序嵌套.

3、C51支持用interrupt n 定义单片机中断函数，using n 确定中断函数所使用的工作寄存器组标准C没有。

绝对地址的访问

1、使用指针

格式： **数据类型［存储类型1］ *[存储类型2］变量名[=&被指向变量名]；**

- 数据类型——被指向变量的数据类型，如int型或char型
- 存储类型1——被指向变量所在的存储区，缺省时由地址赋值关系决定，如data或idata或xdata
- 存储类型2——指针变量所在的存储区，缺省时为编译器默认的存储区，如SMALL默认data区

例： char xdata a = 'A'; //a是存放在xdata区的字符型变量,其值为'A'的ASCII值

char *ptr = &a; //ptr指针变量本身位于SMALL编译模式默认的data存储区里，

//其值为位于xdata存储区里变量a的地址值

2、使用关键字 _at_

格式： 数据类型 [存储器类型] 标识符 _at_ 常数

- 对指定的存储器空间的绝对地址进行定位

例：

```
#define  LENTH 16
unsigned char idata dADDR[LENTH] _at_ 0x30;      //在片内0030H处定义了16字节数组空间。
unsigned char xdata xADDR[LENTH] _at_ 0x3000;    //在片外3000H处定义了16字节数组空间。
```

使用_at_定义的变量只能是全局变量。

绝对地址的访问

3、使用C51运行库中的预定义宏

源程序开头必须包含 **#include <absacc.h>**一行。

```
/*-------------------------------
ABSACC.H

Direct access to 8051, extended 8
Copyright (c) 1988-2002 Keil Elek
All rights reserved.
-------------------------------

#ifndef __ABSACC_H__
#define __ABSACC_H__

#define CBYTE ((unsigned char volatile code  *) 0)
#define DBYTE ((unsigned char volatile data  *) 0)
#define PBYTE ((unsigned char volatile pdata *) 0)
#define XBYTE ((unsigned char volatile xdata *) 0)

#define CWORD ((unsigned int volatile code  *) 0)
#define DWORD ((unsigned int volatile data  *) 0)
#define PWORD ((unsigned int volatile pdata *) 0)
#define XWORD ((unsigned int volatile xdata *) 0)
```

- CBYTE、DBYTE、PBYTE、XBYTE
  用于字节型对code区，data区，pdata区，xdata区的访问
- CWORD、DWORD、PWORD、XWORD
  用于字型对code区，data区，pdata区，xdata区的访问

例如：

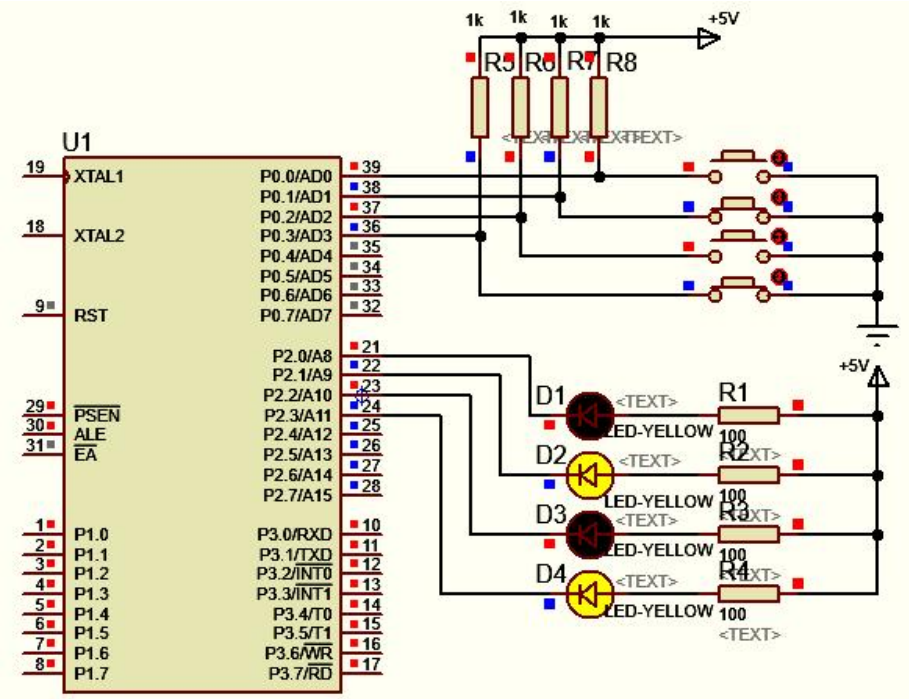
```
uc_var1 = XBYTE [0x0002];
/*将外部RAM 0002H地址的1个字节单元的内容赋给变量*/

ui_var1 = XWORD [0x0002];
/*将外部RAM 0002H和0003H两个地址的2个字节单元的内容赋给变量*/
```

用volatile声明即每次都要重新读取原始内存地址的内容，而不是直接使用保存在寄存器里的值。

- **例1** 独立按键识别，要求开机时LED全熄，然后根据按键动作使相应灯亮，并将亮灯状态保持到按压其它键时为止。

```c
#include <REG51.H>

void main() {
    char key = 0;          //定义按键变量
    P2 = 0xff;             //初始状态为灯全灭

    while(1){
        P0  = 0xff;        //读P0口之前置1
        key = P0;          //读按键状态
        P2  = key;         //显示按键状态
    }
}
```

```c
#include <REG51.H>

void main() {
    char key = 0;          //定义按键变量
    P2 = 0xff;             //初始状态为灯全灭

    while(1){
        P0  = P0 | 0x0f;   //读P0口低4位之前置1
        key = P0 & 0x0f;   //读P0口按键状态高4位清零
        // if ( ~key != 0 )
        P2 = key;          //有键按下，显示按键状态
    }
}
```



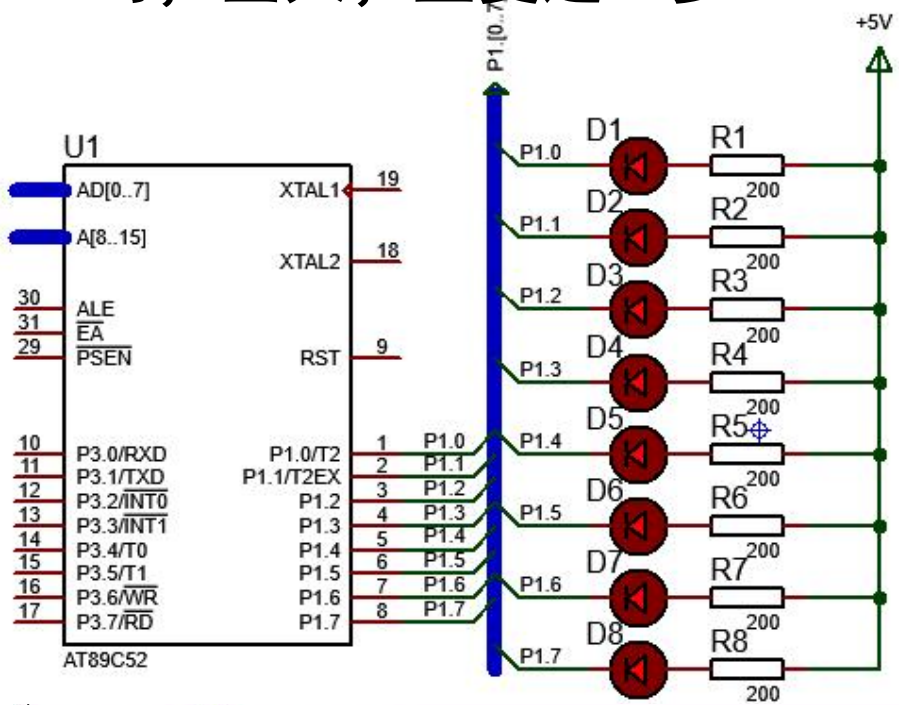
基本输入输出单元与编程

例2 流水灯。要求编写程序实现：

（1）全灭全亮1次，再隔一交替灭亮1次；

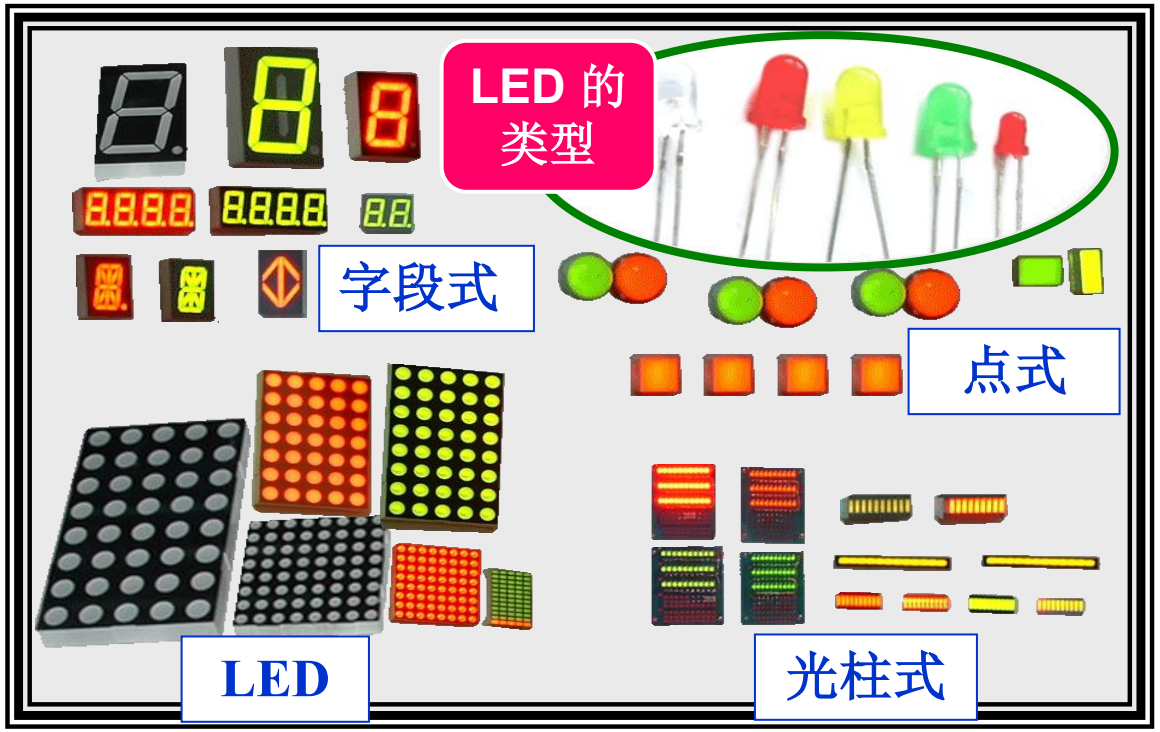
（2）从下往上依次点亮一个LED，当点亮所有LED时，全灭；重复这一步。

```
#include <REG52.H>

#define DELAY_TIME 400 //延时
#define LED_PORT1 P1  //用P1口驱动灯，低亮，高灭

void time(unsigned int ucMs);/* 延时单位：毫秒 */
void main(void)
{
    unsigned char ucTimes;


     LED_PORT1=0xff;time(DELAY_TIME);//全灭
     LED_PORT1=0;time(DELAY_TIME);    //全亮

     LED_PORT1=0x55;time(DELAY_TIME);//隔一个点亮
     LED_PORT1=0xaa;time(DELAY_TIME);//交换

     while(1)
    {
     LED_PORT1=0xff;//灭P1口灯
     time(DELAY_TIME);

      //依次点亮LED
     for(ucTimes=0;ucTimes<8;ucTimes++){//循环点亮P1口灯

         LED_PORT1 = 0xfe<<ucTimes;time(DELAY_TIME);//从上往下依次亮灯
         //LED_PORT1 = 0x7f>>ucTimes;time(DELAY_TIME); //从下往上依次亮灯
         //LED_PORT1 = LED_PORT1 - (0x80>>ucTimes);time(DELAY_TIME);//从下往上依次亮灯
         }
     }
}
```




```
/***********延时***************/
void time(unsigned int ucMs)//延时单位：ms
{
    #define DELAYTIMES 239
    unsigned char  ucCounter;  // 延时设定的循环次数

    while (ucMs!=0) {
        for (ucCounter=0; ucCounter<DELAYTIMES; ucCounter++){  }//延时
        ucMs--;
        }
}
```

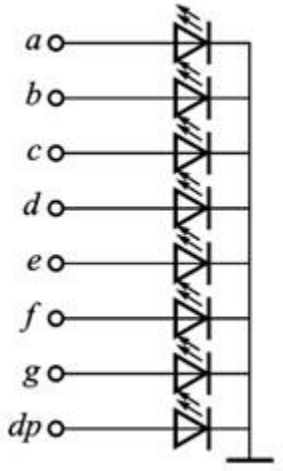
LED数码管原理与编程



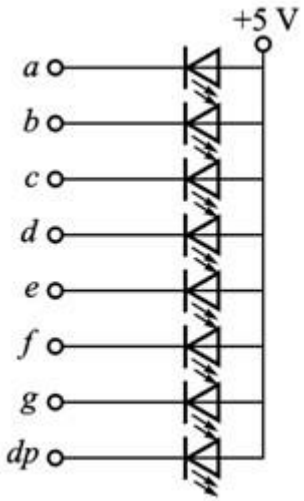
八段数码管的结构



八段数码管



共阴极



共阳极

八段LED数码管如图，习惯上是**以“a”段对应段码字节的最低位，**正好是**一个字节**。

不同的符号或数字，需要相应的段码即**字型码**。

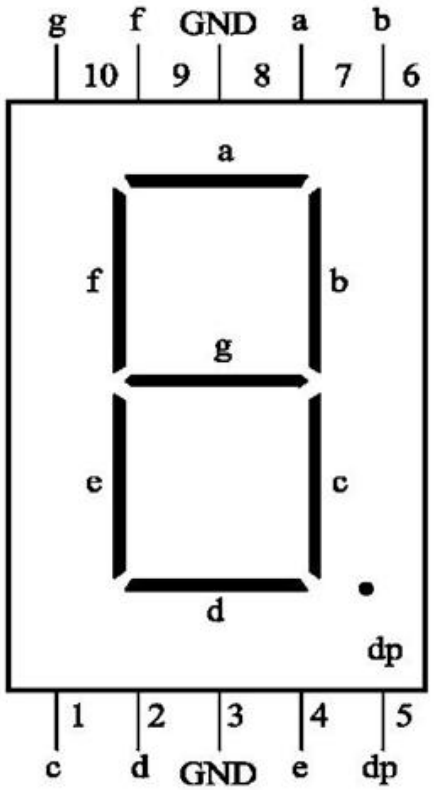


表 10-1 段码与字节中各位对应关系

代码位	D7	D6	D5	D4	D3	D2	D1	D0
显示段	dp	g	f	e	d	c	b	a

显示 0　　a, b, c, d, e, f被点亮

共阴极段选码(高电平点亮)：3FH

Dp	g	f	e	d	c	b	a
0	0	1	1	1	1	1	1

共阳极段选码(低电平点亮)：C0H

Dp	g	f	e	d	c	b	a
1	1	0	0	0	0	0	0

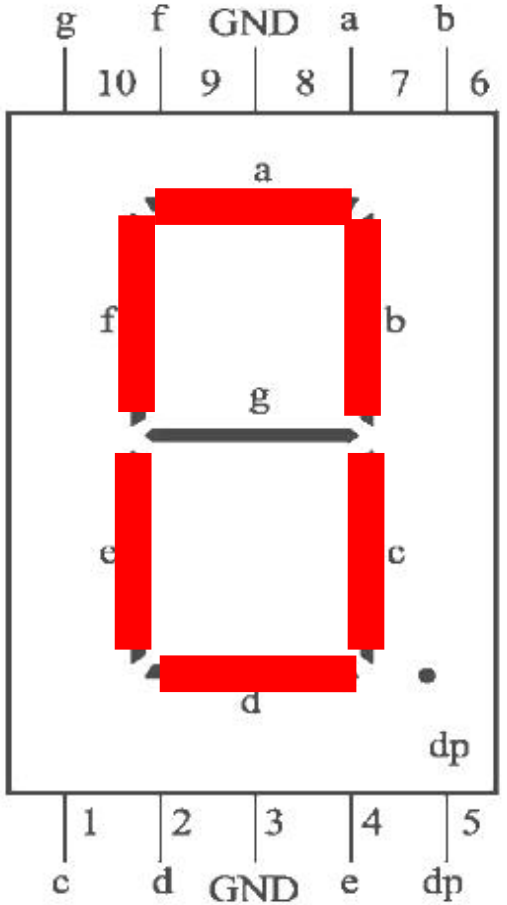


表 10-1　段码与字节中各位对应关系

代码位	D7	D6	D5	D4	D3	D2	D1	D0
显示段	dp	g	f	e	d	c	b	a

表 10-2　8 段 LED 段码

显示字符	共阴极段码	共阳极段码	显示字符	共阴极段码	共阳极段码
0	3FH	C0H	C	39H	C6H
1	06H	F9H	d	5EH	A1H
2	5BH	A4H	E	79H	86H
3	4FH	B0H	F	71H	8EH
4	66H	99H	P	73H	8CH
5	6DH	92H	U	3EH	C1H
6	7DH	82H	T	31H	CEH
7	07H	F8H	y	6EH	91H
8	7FH	80H	H	76H	89H
9	6FH	90H	L	38H	C7H
A	77FH	88H	“灭”	00H	FFH
b	7CH	83H	…	…	…

例3 计数显示器，统计按键次数并以十进制形式显示，按键次数大于99后重新由0开始计数。

将数码管段码值放入数组中，要取用的段码值与数组下标对应。

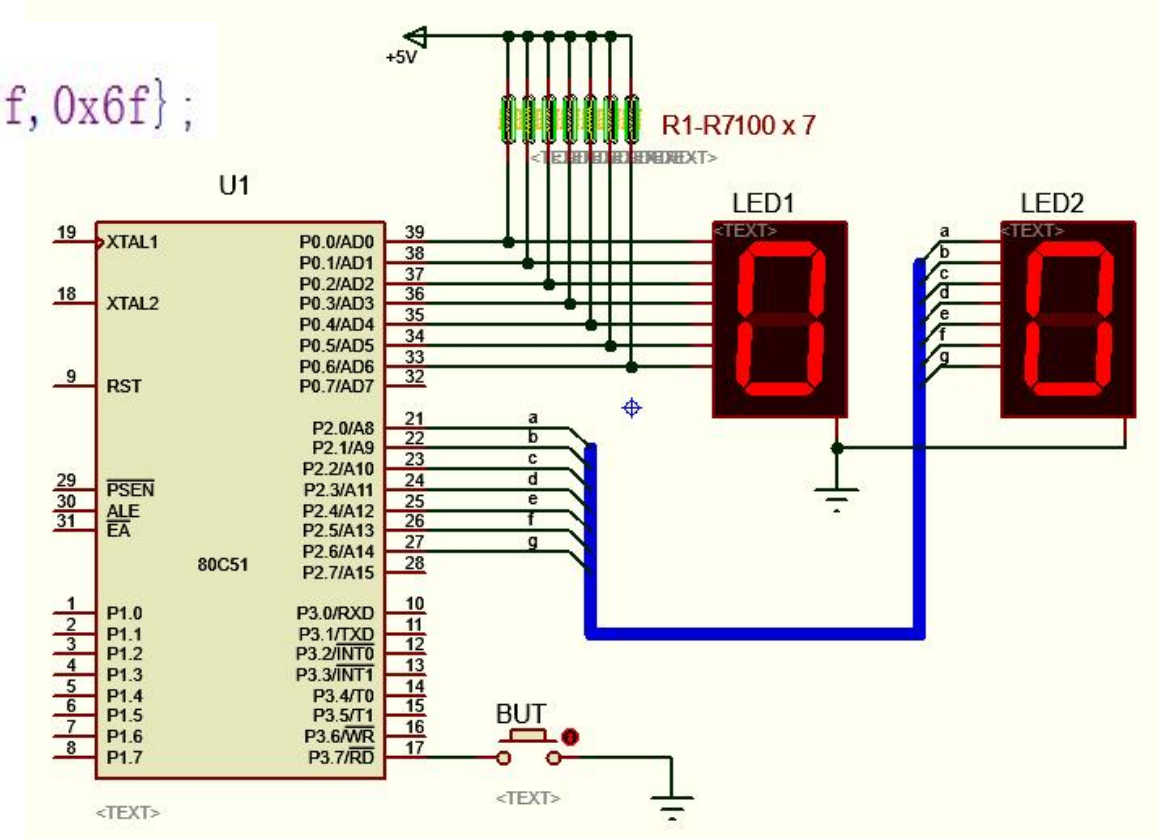
```
unsigned char code table[]=
    {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
```

分析：

- 初始 P2 = P0 = table[0]；//即显示全0
- 如何将计数值拆成两个独立的数显示？

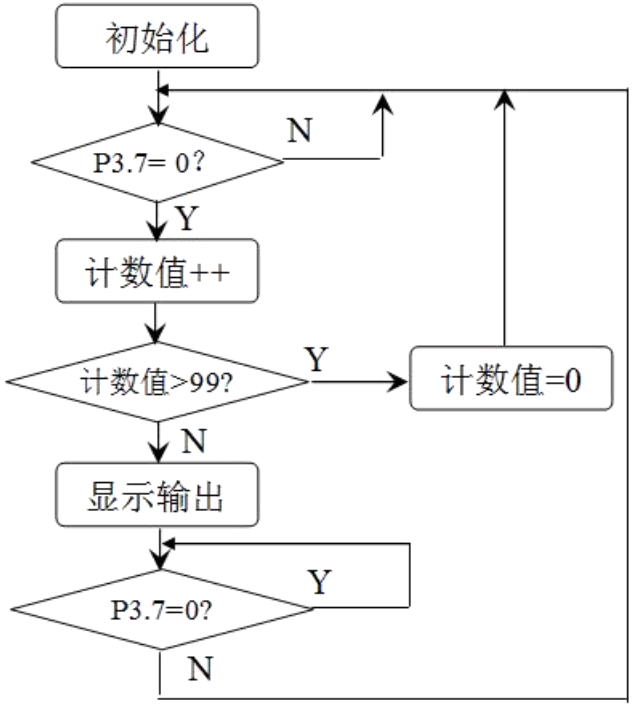
  整除十运算（/10）→十位;P0=table[count/10]

  取模运算（%10）→个位; P2= table[count%10]



问题：按键压下时可能被连续计数，如何解决？即如何避免连续计数?

```
#include <reg51.H>
sbit P3_7=P3^7;                          //定义计数器端口
unsigned char count =0;                  //定义计数器
unsigned char code table[]=
    {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};

void main(void) {
    P0=P2=table[0];                      //显示初值00
    while(1) {
        if(P3_7==0){                     //检测按键是否压下
            count++;                     //计数器增1
            if(count==100) count=0;      //判断循环是否超限
            P0=table[count/10];          //十位输出显示
            P2=table[count%10];          //个位输出显示
            while(P3_7==0);              //等待按键抬起，防止连续计数
        }
    }
}
```



第四讲-1 单片机汇编&C51程序结构 重点

重点：

- 汇编指令与伪指令的作用和本质区别
- 常用伪指令：
  - ORG、END、EQU、BIT
  - DB、DW、DS

- C51的函数特点
- C51的绝对地址访问

- 简单人机交互程序